Programación Concurrente
Prácticas 1, 2 y 3 (r1463)

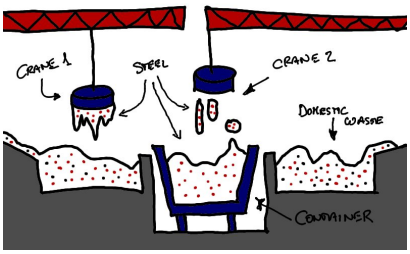
Dpto. de Lenguajes, Sistemas Informáticos e Ing. de Software Convocatoria de Febrero 2009/2010

Normas

- La fecha límite de entrega de la práctica 1 es el **15 de enero de 2010** a las 21:00.
- La fecha límite de entrega de la práctica 2 es el **22 de enero de 2010** a las 21:00.
- La fecha límite de entrega de la práctica 3 es el **29 de enero de 2010** a las 21:00.
- Planeamos publicar los días **28 de diciembre** y **12 de enero** (ambas fechas aproximadas) el estado de las prácticas recibidas hasta ese momento.
- Deberá mencionarse explícitamente el uso de recursos (código, algoritmos específicos, esquemas de implementación, etc.) que no hayan sido desarrollados por el alumno o proporcionados como parte de asignaturas de la carrera. La falta de esta mención podrá impedir aprobar las prácticas.
- Os recordamos que **todas** las prácticas entregadas pasan por un proceso automático de detección de copias. Los involucrados en la copia de una práctica corren el riesgo de suspenderlas para todo el año académico en curso, independientemente de la calidad o adecuación del código.

1. Planta de reciclado

En una planta de reciclado (ver figura), se recupera el acero de entre la chatarra mediante una serie de grúas equipadas con fuertes electroimanes. Estas grúas se encargan luego de depositar el acero en un contenedor hasta que está más o menos lleno.



La grúa es accionada gracias a una librería, parte de cuya interfaz mostramos a continuación:

```
package Gruas is
   MAX_GRUAS : constant Positive := 5;
   subtype Id_Grua is Positive range 1 .. MAX_GRUAS;
   MIN_P_GRUA : constant Positive := 1000;
   MAX_P_GRUA : constant Positive := 1500;
```

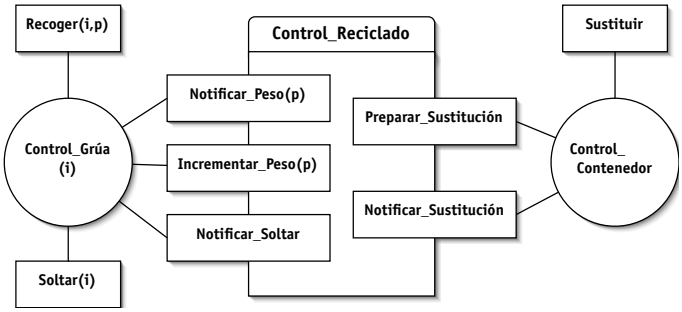


Figura 1: Estructura de procesos

```
   subtype Peso is Positive range MIN_P_GRUA .. MAX_P_GRUA;
   --  Recoge metal e informa de su peso
   procedure Recoger (N : in Id_Grua; P : out Peso);
   --  Mueve la grua hasta el punto de descarga y
   --  desactiva el electroiman
   procedure Soltar (I : Id_Grua);
end Gruas;
```

De igual manera, se dispone de una librería para controlar el desplazamiento del contenedor:

```
package Contenedor is
   MAX_P_CONTENEDOR : constant Natural := 20000;
   --  Sustituye el contenedor actual con otro vacio
   procedure Sustituir;
end Contenedor;
```

Queremos implementar un sistema concurrente para controlar las grúas y el contenedor de manera que no se exceda la capacidad de los contenedores, y sustituir contenedores llenos por otros vacíos, asegurándonos de que nunca se intenta depositar metal en la zona del contenedor mientras éste está siendo reemplazado. Para simplificar el problema supondremos que hay espacio suficiente para que unas grúas no se estorben físicamente con otras.

2. Diseño

Tendremos un proceso para controlar cada grúa y otro más para manejar el reemplazo de contenedores. La comunicación y sincronización es responsabilidad de un gestor. La arquitectura del sistema se muestra en la figura 1.

La idea es que el estado interno del recurso compartido sea suficientemente rico como para determinar cuándo hay que solicitar la sustitución del contenedor y sincronizar a los procesos que controlan las grúas con esa circunstancia.

Más concretamente, se dispondrá de un *estado de reposición* que puede tomar uno de tres valores: *listo*, que quiere decir que el contenedor admite más carga; *sustituible*, que indica que al menos una de las grúas lleva más carga de la que cabe en el contenedor, por lo que dicha grúa solicita un contenedor nuevo, y *sustituyendo*, estado en el cual no se debe depositar carga ya que el contenedor puede estar siendo reemplazado.

Obsérvese que incluso en el estado *sustituible* es posible que una grúa distinta a la que ha solicitado la sustitución del contenedor deposite una cantidad menor en el contenedor.

La especificación del gestor se encuentra en la figura 2.

C-TADSOL Control_Reciclado
OPERACIONES
ACCIÓN Notificar_Peso: *Control_Reciclado [io]* × *Peso[i]*
ACCIÓN Incrementar_Peso: *Control_Reciclado [io]* × *Peso[i]*
ACCIÓN Notificar_Soltar: *Control_Reciclado [io]*
ACCIÓN Preparar_Sustitución: *Control_Reciclado [io]*
ACCIÓN Notificar_Sustitución: *Control_Reciclado [io]*

SEMÁNTICA
DOMINIO:
TIPO: $Control\_Reciclado = (peso: \mathbb{N} \times estado: Tipo\_Estado \times accediendo: \mathbb{N})$
$Tipo\_Estado = listo \mid sustituible \mid sustituyendo$
$Peso = MIN\_P\_GRUA \,..\, MAX\_P\_GRUA$
INICIAL(r): r = (0, listo, 0)
INVARIANTE: $\forall r \in Control\_Reciclado \bullet r.peso \leq MAX\_P\_CONTENEDOR \wedge r.accediendo \leq MAX\_GRUAS$

CPRE: $r.estado \neq sustituyendo$
Notificar_Peso(r, p)
POST: $r^{sal}.peso = r^{ent}.peso \wedge r^{sal}.accediendo = r^{ent}.accediendo \wedge$
$r^{ent}.peso + p > MAX\_P\_CONTENEDOR \rightarrow r^{sal}.estado = sustituible \wedge$
$r^{ent}.peso + p \leq MAX\_P\_CONTENEDOR \rightarrow r^{sal}.estado = listo$

PRE: $p \leq MAX\_P\_GRUA$
CPRE: $r.peso + p \leq MAX\_P\_CONTENEDOR \wedge r.estado \neq sustituyendo$
Incrementar_Peso(r, p)
POST: $r^{ent} = (peso, e, a) \wedge r^{sal} = (peso + p, e, a + 1)$

CPRE: Cierto
Notificar_Soltar(r)
POST: $r^{sal} = (r^{ent}.peso, r^{ent}.estado, r^{ent}.accediendo - 1)$

CPRE: $r = (\_, sustituible, 0)$
Preparar_Sustitución(r)
POST: $r^{sal} = (r^{ent}.peso, sustituyendo, 0)$

PRE: $r = (\_, sustituyendo, 0)$
CPRE: Cierto
Notificar_Sustitución(r)
POST: $r^{sal} = (0, listo, 0)$

Figura 2: CTAD del controlador de la planta de reciclado.

3. Trabajo a realizar

3.1. Primera práctica

La entrega de la **primera práctica** constará de una implementación del recurso en Ada 95 usando objetos protegidos, basada en la especificación dada en el enunciado. La implementación a realizar debe estar contenida en un fichero llamado `control_reciclado.adb` (ver el apartado 4).

3.2. Segunda práctica

La entrega de la **segunda práctica** constará de una implementación del recurso compartido en Ada 95 mediante *rendez-vous* y/o paso de mensajes síncrono y basada en la especificación entregada en el enunciado. La implementación deberá estar contenida en un fichero llamado `control_reciclado.adb` (ver el apartado 4).

3.3. Tercera práctica

Habréis observado que el estado *sustituible* es reversible, es decir, sólo indica que una grúa necesita un contenedor vacío, pero puede llegar otra grúa que borre esa solicitud, cambiando el estado a *listo*, lo que le permitiría soltar su carga. Esto, en principio, está pensado para favorecer el llenado de los contenedores al máximo.

No obstante, también puede suceder que un contenedor sea sustituido sin dar tiempo a que otras grúas lo llenen aún más. Una posible variación consistiría en hacer que el reemplazo por un contenedor nuevo se produjese sólo cuando ninguna de las grúas sea capaz de depositar carga adicional.

Se os pide cambiar la definición de los procesos y / o el recurso (pueden ser ambos) para que el sistema funcione de acuerdo con esta idea.

La entrega de esta **tercera práctica** constará de una memoria con un nuevo diseño que debe cumplir en lo posible los mismos requisitos que se tenían para el problema inicial. En esta entrega deben incluirse la especificación formal del recurso necesario para la sincronización de los procesos y el código de los procesos en función de la interfaz del recurso especificado. La entrega de la práctica se hará de forma **electrónica** en formato PDF.

4. Información general

El texto de estas prácticas se encuentra en

`http://moodle.upm.es/`

La entrega del código y la memoria se realizará **vía WWW** en la dirección

`http://lml.ls.fi.upm.es/entrega/`

El código que finalmente entreguéis (tanto para objetos protegidos como para paso de mensajes) no debe realizar **ninguna** operación de entrada / salida.

Para facilitar la realización de la práctica están disponibles en `http://moodle.upm.es/` varias unidades de compilación:

- `planta.adb`: programa principal que crea el recurso compartido y lanza las tareas de control.

- `gruas.ad?` y `contenedor.ad?`.
- `basicos.ads`: paquete que contiene definiciones de tipos y constantes necesarias en la implementación. Podéis variarlas con objeto de cambiar el comportamiento del sistema.

Por supuesto durante el desarrollo podéis cambiar el código que os entreguemos para hacer diferentes pruebas, depuración, etc., pero el código que entreguéis debe poder compilarse y ejecutar sin errores junto con el resto de los paquetes entregados **sin modificar estos últimos**. Podéis utilizar las librerías auxiliares que estén disponibles en `http://moodle.upm.es/` en la asignatura de Programación Concurrente. Si os veis en la necesidad de usar algún otro paquete adicional que no venga con GNAT y que no os estemos proporcionando, hacédnoslo saber con antelación suficiente para evaluar la petición.

El programa de recepción de prácticas podrá rechazar entregas que:

- Tengan errores de compilación.
- Utilicen otras librerías o paquetes aparte de los estándar de Ada y los que se han mencionado anteriormente.
- No estén suficientemente comentadas. Alrededor de un tercio de las líneas deben ser comentarios **significativos**. No se tomarán en consideración para su evaluación prácticas que tengan comentarios ficticios con el único propósito de rellenar espacio.
- Estén escritas con un estilo incorrecto: los programas se compilarán con el compilador GNAT y las opciones `-gnaty -gnata`. Recomendamos, por tanto, usar estas opciones para desarrollar las prácticas.
- No superen unas pruebas mínimas de ejecución, similares a las que tenéis en el programa de simulación que os entregamos.